

Specifying Efficient Recognizers for Sketch-Based Rendering

Dan Xiao^(✉)

Shenzhen Polytechnic, Xili Lake, Nanshan District, Shenzhen 518055
Guangdong, China
xiaodanl0@szpt.edu.cn

Abstract. Compared to free sketch, gesture-based sketch recognition can achieve high accuracy by requiring the user to learn a particular drawing style in order for shapes to be recognized. In this case, choosing an appropriate classifier is quite critical. This paper compared three different algorithms for labeling each drawn stroke as being a particular component in the generic model. Our statistic shows that K-means classifier yields better results than the other two and we test that by applying this classifier to rocket sketches.

Keywords: Sketch recognition · Sketch-based rendering

1 Introduction

Because of the increasing use of Table PCs, sketch recognition has become one popular forms of human-interaction. Its techniques have generally fallen into two camps, free sketch recognition and gesture-based. Free-sketch recognition allows users to draw shapes as they would naturally, but most current techniques have low accuracies or require significant domain-level adjustment to make them usable. On the other hand, gesture-based require the users to learn a particular drawing style in order for shapes to be recognized and in this way, it can achieve higher accuracies [1].

Some gesture-based recognition techniques build 2D templates for matching. 3D SKETCH [2] is a system which limits the domain of renderable sketches and prepares a template for interpreting sketches. As long as a sketch can be matched to such a template, the system can reconstruct a mesh model from the sketch. Bartelt et al. present an approach to use digital whiteboards to transform free hand sketches in formal models and back again. This approach implemented a collaborate sketch and modeling infrastructure [3]. Cherlin et al. introduced two parametric surfaces, rotational and cross sectional blending, that are inspired by this illustration technique. They also described orthogonal deformation and cross sectional oversketching as editing tools to complement the modeling techniques [4]. Different methods for sketch recognition are explored. Deufemia et al. proposed a two-stage method which combines the use of a discriminative model and a distance-based clustering algorithm for recognizing sketched symbols [5]. Arandjelovic and Sezgin fused an image-based method with a time-based method to form an effective recognizer in an attempt to combine the knowledge of how objects look (image data) with the knowledge of how they are

drawn (temporal data) [6]. Kim et al. presented a curvature estimation method which applies new curvature metrics on corner finding based on local shape information. This method can therefore estimate curvature on-the-fly while user is drawing on a pen-input display [7].

Shape retrieval requires stroke recognition as well. A system called PaleoSketch [8] is able to recognize a larger number of primitive shapes and complex shapes consisting of lines and curves, and at the same time still maintains high recognition accuracy. Eitz et al. developed a system for 3D object retrieval which takes sketched feature lines as input with a targeted feature transform based on Gabor filters [9]. Furthermore, the potential semantic meanings of the user sketch are widely being researched and 3D models are then matched to 2D within the predicted categories [10]. Ghorbel et al. proposed an analyzer based on a competitive breadth-first exploration of the analysis tree. Several possible hypotheses of recognition are then evaluated in a dynamic local context of document [11].

The rest of this paper is organized as follows. Section 2 introduces the whole process for 2D template matching and 3D model rendering. Section 3 compares 3 different classifiers on their efficiency of stroke recognition. Section 4 presents results. Conclusions and future work are discussed in Sect. 5.

2 Sketch Recognition

For user learning sketch rendering, the assumption we made is that the user choose the class of object they want to draw. The benefit of doing that is we can then apply corresponding 2D template to those user drawn graphs and then render them. Figure 1 shows the 2D template of a rocket which consists of five strokes. Five components for building this template are illustrated in Table 1. For a free drawn graph of rocket, once its five components are recognized, it can then rendered with a 2D template. In order to recognize those components, we have to find an efficient classifier.

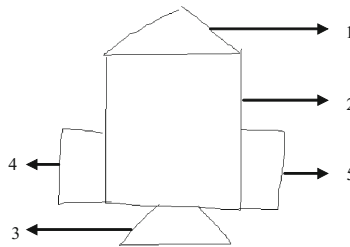


Fig. 1. Illustration of rocket components.

Table 1. Description of components for rocket

Order	Components	Shapes
1	head	triangle
2	body	open triangle
3	tail	open triangle
4	left wing	open triangle
5	right wing	open triangle

Each gesture is defined as an array g of coordinate values and time.

$$g_p = (x_p, y_p, t_p) \quad 0 \leq p < P$$

The gesture recognition problem is stated as follows. There is a gesture set with C classes. Given a gesture, we need to determine to which class it belongs. This is done by first extracting a limited set of features. Features are extracted from the gesture g and are used as the input for classification. The feature vector $f = [f_1 \dots f_p]$ is taken as the training data for classification.

A single stroke can be characterized by a set of 11 geometric and 2 dynamic features. This is the set of features we shall use for testing several classification algorithms. The description of features is:

- Feature 1 (f_1): the cosine of the initial angle of the gesture.
- Feature 2 (f_2): the sine value of the initial angle of the gesture.
- Feature 3 (f_3): the length of the bounding box diagonal.
- Feature 4 (f_4): the angle of the bounding box diagonal.
- Feature 5 (f_5): the distance between the first and the last point.
- Feature 6 (f_6): the cosine of the angle between the first and last point.
- Feature 7 (f_7): the sine of the angle between the first and last point.
- Feature 8 (f_8): the total gesture length.
- Feature 9 (f_9): the total angle traversed.
- Feature 10 (f_{10}): the sum of the absolute value of the angle at each mouse point.
- Feature 11 (f_{11}): the sum of the squared value of those angles.
- Feature 12 (f_{12}): the maximum speed (squared) of the gesture.
- Feature 13 (f_{13}): the duration of the gesture.

3 Methods for Stroke Classification

Three machine learning classification schemes are used in order to determine which one works best in our case: least squares method, k-means clustering and expectation-maximization method.

3.1 Least Squares Method

Rubine uses a linear machine algorithm for strokes classification, which is a kind of least square approach [12].

Given C as the number of total classes and F is the number of features, each gesture class c has parameters ω_{ci} for $0 \leq i \leq F$. The evaluation v_c is calculated as follows:

$$v_c = \omega_{c0} + \sum_{i=1}^F \omega_{ci} f_i \quad 0 \leq c < C$$

The class of a gesture g is the c which maximizes v_c .

During the training process, we need to determine the weights ω_{ci} from the example gestures. Let f_{cei} be the feature of the example of gesture class c , $0 \leq e \leq E_c$, where E_c is the number of training examples of class c .

The sample estimate of the mean feature vector per class \bar{f}_c is defined as:

$$\bar{f}_{ci} = \frac{1}{E_c} \sum_{e=0}^{E_c-1} f_{cei}$$

The sample estimate of the covariance matrix of class c , Σ_{cij} , is computed as:

$$\Sigma_{cij} = \sum_{e=0}^{E_c-1} (f_{cei} - \bar{f}_{ci})(f_{cej} - \bar{f}_{cj})$$

The Σ_{cij} are averaged to yield Σ_{ij} , an estimate of the common covariance matrix.

$$\Sigma_{ij} = \frac{\sum_{c=0}^{C-1} \Sigma_{cij}}{-C + \sum_{c=0}^{C-1} E_c}$$

We can invert the common covariance matrix and use it to get the weights ω_{ci} as follows:

$$\omega_{cj} = \sum_{i=1}^F (\Sigma^{-1})_{ij} \bar{f}_{ci} \quad 1 \leq j \leq F$$

$$\omega_{c0} = -\frac{1}{2} \sum_{i=1}^F \omega_{ci} \bar{f}_{ci}$$

3.2 K-Means Method

We use K-means clustering introduced in [13] for classification. In K-means clustering, K, needs to be determined at the onset. The goal is to divide the objects into K clusters such that some metric relative to the centroids of the clusters is minimized. Various metrics related to the centroids can be minimized, including:

The maximum distance to its centroid for any object.

- The sum of the average distance to the centroids over all clusters.
- The sum of the variance over all clusters.
- The total distance between all objects and their centroids.
- The metric to minimize and the choice of a distance measure will determine the shape of the optimum clusters.

Two different algorithms are available to search for the optimum set of clusters. In the first procedure, the objects are randomly assigned to one of the K clusters. Once this is done, the position of the K centroids is determined, as is the value of the metric to minimize. A global optimization method is then used to reassign some of the objects to different clusters. New centroids are determined, as is the metric to minimize. This procedure is continued until the optimum assignment of objects to clusters is found.

In the second procedure for K-means clustering, placement of the K centroids can be done by the following procedure. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.

1. Assign each object to the group that has the closest centroid.
2. When all objects have been assigned, recalculate the positions of the K centroids.
3. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

A global optimization method is then used to move the position of one or more of the centroids. The above procedure is repeated and new metric value is determined. The object is to move the centroids into a position such that an optimum separation of objects into groups occurs.

For K-Means clustering, it is necessary to calculate a “distance” between either two objects (one of which may be a cluster seed point) or an object and a group centroid. In this discussion, it is assumed that each object is described by an array of real-valued metrics.

Since we need to minimize the largest or average distance, or the variance, during clustering, it is important that each metric contribute equally to the total distance. In other words, if one metric spans the range [0.0,0.5] and another spans [0.0,100.0], the maximum deviation in the first would have little effect on the total distance, while even a modest separation in the second would have a much larger effect. To remove this dependency on the range spanned by each metric, it is important to first standardize the values. This means that each metric, when compared over the full set of objects should have a mean of 0.0 and a variance (or standard deviation) of 1.0. For each metric, the following steps should be taken to standardize each metric describing the objects.

1. Sum the values of the metric over all objects and divide the sum by the number of objects.
2. Subtract this average value from the metric in all objects.
3. Sum the square of these new values over all objects, divide the sum by the total number of objects, and take its square-root. This is the standard deviation of the new values.
4. Divide the metric by the standard deviation in each object.

3.3 Expectation-Maximization Method

This method supposes that the stroke features have normal Gaussian distributions. The EM method can be applied to build a gesture classifier. The detailed steps are given by Jordan [14]:

1. Initialize.
2. E Step: At iteration t , compute the expectation of the indicators for each i and c :

$$\zeta_{ic} = \frac{p(c)N(x_i|\mu_c, \Sigma_c)}{\sum_{c'} p(c')N(x_i|\mu_{c'}, \Sigma_{c'})}$$

and normalize it.

3. M Step: Update the parameters $p(c)$, μ_c , Σ_c .

$$\mu_c = \frac{\sum_{i=1}^n \zeta_{ic} x_i}{\sum_{i=1}^n \zeta_{ic}}$$

$$\Sigma_c = \frac{\sum_{i=1}^n \zeta_{ic} (x_i - \mu_c)(x_i - \mu_c)'}{\sum_{i=1}^n \zeta_{ic}}$$

$$p(c) = \frac{1}{n} \sum_{i=1}^n \zeta_{ic}$$

The EM method is used to find the parameters for the normal distribution of each class and then apply this distribution to classify new, unseen feature vectors. The distribution with the highest probability is defined to be the class to which the gesture belongs.

3.4 Classifier Comparison

The gestures used for evaluation of the three classification algorithms are shown in Fig. 2 In this figure, four different gesture styles of each class are given.

Performance is evaluated on 3 gesture classes. Figure 3 shows the results using the least square method. The plot demonstrates the recognition rate as a function of the number of training examples per class for evaluation gestures. One line is for 2 classes and the other one is for 3 classes. Figure 4 illustrates the results with k-means. The results when applying EM are shown in Fig. 5.

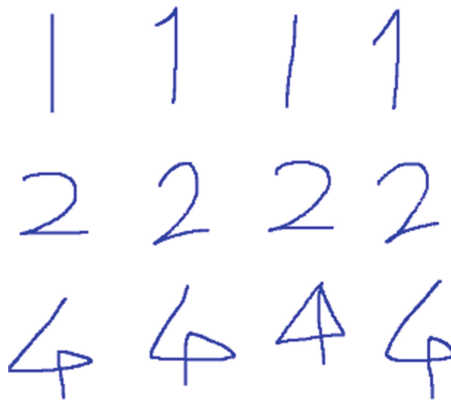


Fig. 2. Examples from the gesture set used for evaluation.

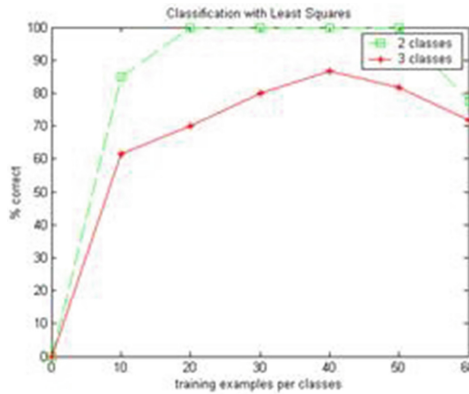


Fig. 3. Recognition rate vs. training size for least square method.

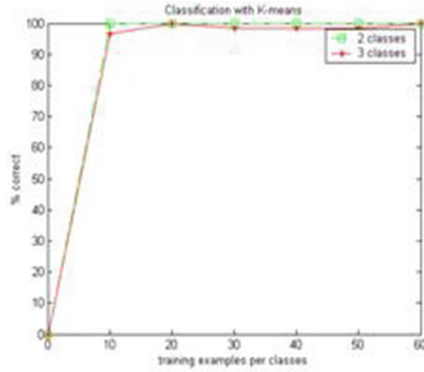


Fig. 4. Recognition rate vs. training size for K-means method.

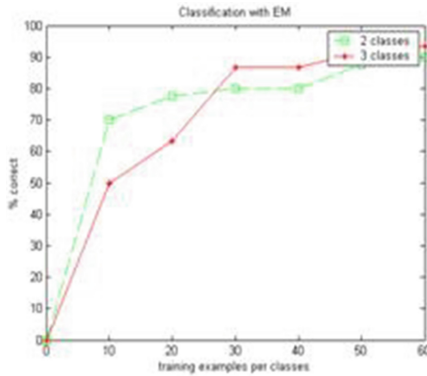


Fig. 5. Recognition rate vs. training size for EM method.

From the above three figures, we can see that the K-means method gives the best performance. For the K-means method, in the cases where 3 gesture classes are recognized by a classifier trained with 10 or more examples per class, at least 96 % of the test gestures are classified correctly. When there are only 2 gesture classes, the classifier trained with 10 or more examples per class yields 100 % correct classification.

The linear machine classifier used by Rubine that is trained with 10 or more examples can recognize 3 gesture classes with correctness >60 %, and for 2 gesture classes, the adequacy is >85 %. This method gives better performance when there are 40 training examples per class, but the correctness falls after increasing the number of examples.

The recognition adequacy of EM method is low when the number of training examples is less than 30 per class, however, it increases when more training examples added. The more training examples per class, the higher the correctness at recognizing gestures. The classifier trained with 30 or more examples can recognize 3 gesture classes at above 85 % correctness, and for 2 gesture classes above 80 % correctness. It

is surprising to note that it will give better performance for 3 gesture classes than 2 gesture classes for large number of training examples.

4 Results

Sketch recognition consists of assigning feature labels to individual pen strokes, i.e., identifying the object features they corresponded to. We have chosen K-means method for stroke classification and we will show how we select features and apply this classifier to stroke matching.

Four features as the intrinsic properties of a stroke are defined to interpret one single stroke, including:

- Feature 1 (f_1): $f_1 = \alpha$, α is the angle of the straight line between the curve endpoints with respect to the horizontal.
- Feature 2 (f_2): $f_2 = 1/d$, l is the length of the curve, d is the straight-line distance between the endpoints.
- Feature 3 (f_3): $f_3 = A/d^2$, A is the signed area between the curve and the straight-line.
- Feature 4 (f_4): the duration of drawing.

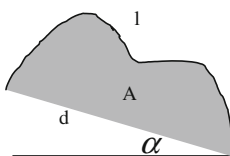


Fig. 6. Key parameters used to computer stroke features.

Figure 6 shows the key parameters used to computer the four features of a stroke. A rocket is described by 5 strokes that can be drawn in any order: head, body, tail, left wing and right wing. After we acquire all the features for each stroke, we use a K-mean classifier to distinguish them. Figure 7 show the 3D rockets rendered from simple sketches.

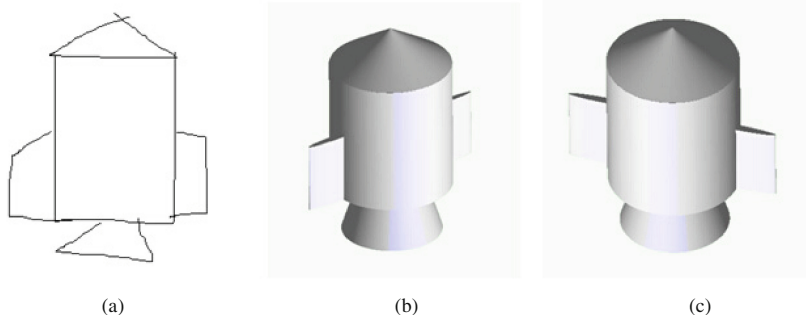


Fig. 7. Rocket modeling (a) original sketch of a rocket. (b) (c) synthesized 3D models, taking from different angles.

The matching process employs a normalization metric that is based on stroke feature vectors, and the search space of possible correspondences is restricted by encoding knowledge about relative part locations into the 2D template. Once every stroke is matched to this 2D template, a 3D object is constructed using a series of data that are extracted from the labeled 2D sketch. We apply our sketch-recognition and modeling algorithms to sketches of rocket and this system allows non-experts to use drawings to quickly create 3D models of specific object classes.

5 Conclusions

In user learning rendering system, a 2D template is applied for certain class of object. Several machine learning methods are tested to find the best-fit classifier for sketch recognition. As a result, K-means classifier gives the best performance and is used to render rocket 3D models. Since the 3D model rendering is based on a certain 2D template, every time a new class of objects is induced in this system, we have to build a new 2D template for it. This will limit the scope of system's application. One possible way to solve this problem is taking on a user learning process to form a 2D template. With this information, the system can create 2D template from simple sketches.

Another important research direction is to improve the recognition algorithm, such as curve features identification. Identify more shapes to be recognized using geometric characteristics or other features. In addition, support multiple-stroke drawing by determining time interval between two continuously drawn strokes may improve the algorithm efficiency.

References

1. Tracy, H., Eoff, B., Paulson, B., Wolin, A., Dahmen, K., Johnston, J., Rajan, P.: Free-sketch recognition: putting the CHI in sketching. In: CHI 2008, Florence, Italy (5 April–10 April 2008)
2. Mitani, J., Suzuki, H., Kimura, F.: 3D sketch: sketch-based model reconstruction and rendering. In: Cugini, U., Wozny, M. (eds.) *From Geometric Modeling to Shape Modeling*. IFIP, vol. 80, pp. 85–98. Springer, Heidelberg (2002)
3. Bartelt, C., Vogel, M., Warnecke, T.: Collaborative creativity: from hand drawn sketches to formal domain specific models and back again. In: *MoRoCo@ ECSCW*, pp. 25–32 (2013)
4. Cherlin, J.J., Samavati, F., Sousa, M.C., Jorge, A.: Sketch-based modeling with few strokes. In: *Proceeding SCCG 2005 Proceedings of the 21st Spring Conference on Computer Graphics*, pp. 137–145. ACM, New York (2005)
5. Deufemia, V., Risi, M., Tortora, G.: Sketched symbol recognition using latent-dynamic conditional random fields and distance-based clustering. *Pattern Recogn.* **47**(3), 1159–1171 (2014)
6. Arandjelovic, R., Sezgin, T.M.: Sketch recognition by fusion of temporal and image-based features. *Pattern Recogn.* **44**(6), 1225–1234 (2011)
7. Kim, D.H., Kim, M.J.: A curvature estimation for pen input segmentation in sketch-based modeling. *Comput. Aided Des.* **38**, 238–248 (2006)

8. Brandon, P., Hammond, T.: PaleoSketch: accurate primitive sketch recognition and beautification. In: Proceedings of the 13th International Conference on Intelligent User Interfaces, pp. 1–10. ACM, New York (2008)
9. Eitz, M., Richter, R., Boubekeur, T., Hildebrand, K., Alexa, M.: Sketch-based shape retrieval. *ACM Trans. Graph.* **31**(4), 31 (2012)
10. Li, B., Lu, Y., Fares, R.: Semantic sketch-based 3D model retrieval. In: 2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW), pp. 1–4. IEEE (2013)
11. Ghorbel, A., Anquetil, E., Camillerapp, J., et al.: IMISketch: an interactive method for sketch recognition. *Pattern Recogn. Lett.* **35**, 78–90 (2014)
12. Rubine, D.: Specifying gestures by example. In: SIGGRAPH 1991, Computer Graphics Proceedings, vol. 25, pp. 329–337 (1991)
13. Moore, A.: K-means and Hierarchical Clustering - Tutorial Slides. <http://www-2.cs.cmu.edu/~awm/tutorials/kmeans.html>
14. Jordan, M.I.: An Introduction to Probabilistic Graphical Models (2004)